Task Scheduling and Data Assignment

Input file: `standard input`
Output file: `standard output`

With the rapid development of chip hardware computing capabilities, hardware architectures are evolving towards more complex multi-core and heterogeneous architectures. In this problem, your goal is to design an efficient task scheduling algorithm together with a memory allocation policy for the software-hardware co-optimization problem that exists in operating systems, distributed systems, and compilers.

Description

There are ℓ tasks that need to be scheduled on n machines with varying processing powers. There are also m disks with varying transmission speeds and storage capacities for storing the data generated by each task after its execution.

Task schedule. A task schedule is an assignment of tasks to machines and disks. More specifically, for each task i, a task schedule specifies three parameters (x_i, y_i, z_i), meaning that i starts at time x_i on machine y_i and stores its data to disk z_i. The earliest starting time of any task is 0.

Task Running Process Here are more details about the running process of a task i when the task schedule is fixed.

- The processing power of machine j is $\mathsf{power}(j)$. Each task i has a size of $\mathsf{size}(i)$, which means it needs $\lceil \mathsf{size}(i)/\mathsf{power}(y_i) \rceil$ units of time to execute when it is run on machine y_i. Here, $\lceil . \rceil$ means rounding up.
- Each disk k has a read/write transmission rate of $\mathsf{speed}(k)$. When a task i finishes its execution, it will generate a piece of data of size $\mathsf{data}(i)$ that needs to be stored on some disk. This storing step will take $\lceil \mathsf{data}(i)/\mathsf{speed}(z_i) \rceil$ units of time if this data is stored on disk z_i. Here, the value of time also needs to be rounded up.
- Before a task i starts its execution, it needs to read the output data of some other tasks. We let $\mathsf{PRED}_{\mathsf{data}}(i)$ denote the set of tasks whose output data are needed by task i. This will take a total time of $\sum_{j \in \mathsf{PRED}_{\mathsf{data}}(i)} \lceil \mathsf{data}(j)/\mathsf{speed}(z_j) \rceil$ and this must be done before task i starts its execution. Note that when calculating the total time, each term needs to be rounded up individually and then summed.

Summarizing the above process, when task i starts, it goes through three phases: reading data from other tasks, executing the task, and storing data. For convenience, we denote the starting time of the three phases as a_i, b_i, c_i and the finishing time of the last phase as d_i. These values are determined as follows.

- $a_i = x_i$
- $b_i = a_i + \sum_{j \in \mathsf{PRED}_{\mathsf{data}}(i)} \lceil \mathsf{data}(j)/\mathsf{speed}(z_j) \rceil$.
- $c_i = b_i + \lceil \mathsf{size}(i)/\mathsf{power}(y_i) \rceil$.
- $d_i = c_i + \lceil \mathsf{data}(i)/\mathsf{speed}(z_i) \rceil$.

In addition, a feasible task schedule should satisfy the following requirements:

- **No preemption**: Each machine executes only one task or transmits one data at a time, and no interruption is allowed during its lifecycle. In other words, for each pair of tasks i and j such that $y_i = y_j$, the two intervals (a_i, d_i) and (a_j, d_j) cannot have any overlaps.

- **Task dependencies**: Each task i has a list of task-dependent tasks, denoted by $\mathsf{PRED}_{\mathsf{task}}(i)$. For each task $j \in \mathsf{PRED}_{\mathsf{task}}(i)$, task i cannot start until j finishes its execution. That is, it must satisfy $a_i \geq c_j$. Note that if i is scheduled on the same machine with j, then it still needs to wait for j to complete storing its data.
- **Data dependencies**: For each task $j \in \mathsf{PRED}_{\mathsf{data}}(i)$, task i cannot start until j finishes storing its data. That is, it must satisfy $a_i \geq d_j$.
- **Affinity of machines**: Each task i has a list of affinitive machines. denoted by A_i, that it can be assigned to. Each task must be scheduled on one of its affinitive machines. That is, it must satisfy $y_i \in A_i$.
- **Disk Capacity**: Each disk k has a storage capacity of $\mathsf{capacity}(k)$. The total size of all data stored on the same disk cannot exceed that disk's capacity. That is, for each disk k it must satisfy $\sum_{i:z_i=k} \mathsf{data}(i) \leq \mathsf{capacity}(k)$.

Given a feasible schedule, the **makespan** of this schedule is the finishing time of the last task. In other words, $\mathsf{makespan} = \max_{1 \leq i \leq \ell} d_i$

Objectives

Your objective is to find a feasible task schedule that minimizes the makespan.

Input

The first line contains an integer ℓ representing the number of tasks. In the following ℓ lines, each line represents a task: The first three integers represent task id i, task size $\mathsf{size}(i)$, and the size of its output data $\mathsf{data}(i)$. All task ids are distinctive integers between 1 and ℓ. The fourth integer k is the number of affinitive machines for this task, and the remaining k integers are the IDs of these affinitive machines and they are distinctive as well.

Next, a line contains an integer n representing the number of machines. In the following n lines, each line has two integers representing machine id j and its power $\mathsf{power}(j)$. All machine ids are distinctive integers between 1 and n.

Next again, a line contains an integer m representing the number of disks. In the following m lines, each line has three integers representing disk id k, its speed $\mathsf{speed}(k)$, and capacity $\mathsf{capacity}(k)$. All disk ids are distinctive integers between 1 and m.

The remaining part describes the two types of dependencies between tasks.

The first line of this part is an integer N representing the number of data dependencies. In the following N lines, each line contains two integers i, j, which means task j is data-dependent on task i.

Next, there is a new line with an integer M representing the number of task dependencies. In the following M lines, each line contains two integers i, j, which means task j is task-dependent on task i.

All numbers on the same line are separated by whitespaces.

Variable constraints

- number of tasks: $6 \leq l \leq 10000$
- number of machines: $1 \leq n \leq 50$
- number of disks: $2 \leq m \leq 30$
- size of task: $10 \leq \mathsf{size}(i) \leq 600$
- size of outputted data: $0 \leq \mathsf{data}(i) \leq 20$
- computing power of machines: $1 \leq \mathsf{power}(i) \leq 20$

- capacity of disks: $1 \leq \mathsf{capacity} \leq 1050000$
- rate of disks: $1 \leq \mathsf{speed} \leq 20$
- all the ids are consequent and starts from one.
- the dependence graph which includes both two types of dependence edge is acyclic.
- size of dependence: $M + N \leq 500000$

Output

You should output ℓ lines describing the task schedule for all tasks. The ith line should contain four integers i, x_i, y_i, z_i, separated by whitespaces, meaning that task i will start at time x_i on machine y_i and store its data on disk z_i.

Example

standard input	standard output
6 1 40 6 2 1 2 2 20 6 2 1 2 3 96 10 2 1 2 4 20 6 2 1 2 5 60 0 2 1 2 6 20 0 1 1 2 1 1 2 2 2 1 1 30 2 2 17 8 1 2 1 3 1 4 2 4 2 5 3 5 3 6 4 6 1 1 5	1 0 2 2 2 23 1 1 3 23 2 2 4 52 1 1 5 79 2 1 6 87 1 2

Note

At time 0, J_1 is scheduled on M_2 and stores its output to D_2, it costs $40/2$ and $6/2$ unit time in executing phase and writing phase, respectively. So, it finishes at time 23. Note that the time cost in the reading phase is zero since it does not data-dependent on any task.

At time 23, J_2 is scheduled on M_1 and reads its data from D_2 and stores its output to D_1. Finally, It costs $6/2 + 20 + 6 = 29$ unit time and it finishes at time 52.

At the same time, J_3 is scheduled on M_2 and reads its data from D_2. And it costs $6/2 + 96/2 + 10/2$ and finishes at 79. Note though J_3 and J_1 are running on the same machine, J_3 still needs to read the J_1's outputted data from disk.

At time 52, J_4 starts on M_1 and reads data of J_1 and J_2, which costs $6/2 + 6/1$ unit time. So the final time cost is $(3 + 6) + 20 + 6$ and finishes at time 87.

At time 79, J_5 starts on M_2 and costs $10/2 + 6$ unit time to read data and another $60/2$ unit time to execute, and does not output any data. So, it finishes at time 120.

At time 87, J_6 starts on M_1 and finishes at time 118.

So, the final makespan is 120 and it is a feasible solution. Let's verify the feasibility of the above solution.

Dependency constraints: In the solution, J_2,J_3 start after J_1 and J_4 starts after J_1,J_2, and so on.

The affinity of machines: In this case, the only restriction of affinity is that task 6 can only run in machine 1. And in this solution, it DOES.

No preemption: it is obvious.

The quota of disk: (1) The tasks storing data in disk 1 are J_2, J_4, J_5, their total data size is $6+6+0 < 30$; (2) The tasks storing data in disk 2 are J_1, J_3, J_6, their total data size is $6 + 10 < 17$.